draft-girod-urn-res-require

Last Version: draft-girod-urn-res-require-00.txt
Date: 13-Jun-1996
Disposition: non-current

```
HTTP/1.1 200 OK
Date: Tue, 09 Apr 2002 00:06:45 GMT
Server: Apache/1.3.20 (Unix)
Last-Modified: Thu, 13 Jun 1996 23:26:00 GMT
ETag: "3ddf7b-1860f-31c0a388"
Accept-Ranges: bytes
Content-Length: 99855
Connection: close
Content-Type: text/plain
```

Internet Draft   Lewis Girod
draft-girod-urn-res-require-00.txt   Karen R. Sollins
                                                MIT LCS
Expires December 13, 1996   June 13, 1996

Requirements for URN Resolution Systems

Status of this draft

Abstract

This paper presents a set of requirements for systems that resolve URNs into hints for locating resources. Hints are a type of metadata that provide information about locating a resource, including but not limited to URLs and pointers to other resolution systems. The requirements fall into three broad areas: usability, security, and evolvability. With those in mind, the paper reviews what we have been able to learn about the NAPTR proposal. The NAPTR proposal has grown out, of a set of several different URN resolution proposals; it uses the existing DNS infrastructure to store and serve resolution information. To this end they introduce a new kind of DNS entry called a NAPTR, or Naming Authority PoinTeR. The review of the NAPTR proposal includes our wishlist for extensions and modifications to it that would add evolutionary paths to the design without sacrificing ease of implementation. The paper then presents a sketch of a more extensive URN resolution system, presented in order to demonstrate the need for evolution in any URN resolution proposal.

1 Introduction

This document sets out some requirements which apply to URN resolution systems. By URN resolution system we mean the whole process of resolution, beginning with a URN with no hints as to how to resolve it, and ending with one or more hints describing with reasonable certainty and possibly with verifiable authenticity the location of the named resource. We also include in such a system the mechanism by which entities publish named resources.

It is important to note that this system is not the only method of resolving URNs. In fact, documents containing URN references would typically be bundled with a collection of hints when they arrive from the server. In most cases these hints would provide a direct resolution; however, in some cases the hints will be out of date. It is these cases, along with the cases where a URN is entered directly by the user, in which the system we are concerned with here is used.

The requirements applying to a URN resolution system center around three important design goals. While it may be neither feasible nor necessary that initial implementations support every requirement, every implementation must support evolution to systems that do support every requirement.

* USABILITY: URNs are long lived identifiers. It is not sufficient for a URN resolution system merely to make it _possible_ for URNs to have long lifespans; a URN resolution system must _encourage_ the maintenance of long-lived names by virtue of its design, performance, and the economic structure it imposes. This is a very broad and openly interpretable requirement.

* SECURITY: Today the amount of information published electronically is growing and it is predicted to continue. An acceptable security model must extend beyond the individual servers that provide such published information. It must support distributed security policies for not only the published information itself but also for hint information. This extends to authenticity, security from unauthorized modification, and privacy.

* EVOLVABILITY: It is imperative that any deployed URN system have extensibility built into the design at many levels of the system. At the lowest levels this means that systems must be designed to be able to take advantage of new or improved transport protocols. At the middleware levels, where this work falls, functionality may be increased or improved over time. Thus within a URN resolution system, there must be a path for evolution of the resolution model. This may include both the co-existence of earlier and newer resolution models, and the eventual phasing out of older schemes gracefully. The Internet is beyond the point where a globally deployed system with broad usage can support a ``flag day'', when a transition occurs abruptly from one model to another. At the highest levels, the level of applications themselves, evolution may place changing requirements on the middleware levels. For example, if applications store, retrieve and exchange files of a limited number of types such as text and binary, then there is no need for an extensible typing system to support them. In contrast, if applications define and exchange objects of new and evolving types, the middleware substrate will be required to support that exchange. We see evolution occurring at all levels of the network.

This paper proceeds as follows. First, we will expand on the set of goals listed above, describing them in more detail. This will be followed with a discussion of two URN resolution system proposals. The first is the NAPTR proposal that comes out of followup discussions from the now defunct URI working group of the IETF and a collection of proposals for URN resolution that were developed in that context. The second is our own proposal. We view the first as an interim solution that does not address all the issues, but can be made at least to support evolution. It has the advantage that it probably has a shorter deployment path. The second addresses more of the issues more effectively, but probably has a longer deployment path, and therefore could be viewed as the planned successor to the NAPTR scheme.

2 The goals and issues

A URN resolution must provide the following components of functionality:

* Publication of pieces of information, which implies some mechanism or set of mechanisms for making the identity of a piece of information known to an audience.

* Hint publication, which implies the ability to make known one or more paths to locating a piece of information

* Hint discovery, which implies the ability to learn hints, by more or less trustworthy mechanisms.

There are some auxiliary functions that are important to identify as well, in particular, authentication and access control of the hint information and pricing mechanisms for storage and management of hint information. If the hint resolution mechanism is to be useful, it must be credible, and in order to achieve that there must be some degree of guarantee that the information stored, managed, and distributed is correct. This will require an authentication and access control mechanism. In addition, there may be problems related to the volume and rate of change of hint information. The more information there is and the more frequently it changes the more difficult it will be to provide a valuable, well functioning service. Hence a pricing mechanism may be needed to put some amount of negative pressure on updates to the central information sources. Both the security and pricing issues will be discussed further below.

2.1 Usability and Feature Set Requirements

Usability can be evaluated from three distinct perspectives: those of a publisher wishing to make a piece of information public, those of a client requesting URN resolution, and those of the provider or manager of resolution information. We will separately address the usability requirements from each of these three perspectives.

2.1.1 The Publisher

The publisher must be able to make URNs known to potential customers. From the perspective of a publisher, it is of primary importance that URNs be correctly and efficiently resolvable by potential clients. Publishers also stand to gain from long-lived URNs, since they increase the chance that references continue to point to their published resources. The publisher must also be able to choose easily among a variety of potential services that might translate URNs to location information. In order to allow for this mobility among resolution services, the architecture for resolution services specified within the IETF should not result in a scenario in which changing from one resolution service to another is an expensive operation.

The publisher should be able to arrange for multiple access points to a published resource. For this to be useful, resolution services should be prepared to provide different resolution or hint information to different clients, based on a variety of information including location and the various access privileges the client might have. For example, companies might arrange for locally replicated copies of popular resources, and would like to provide access to the local copies only for their own employees. This is distinct from access control on the resource as a whole, and may be applied differently to different copies.

The publisher should be able to provide both long and short term information about accessing the resource. Long term information is likely to be such information as the long term location of the resource or the location or identity of a resolution service with which the publisher has a long term relationship. One can imagine that the arrangement with such a long term ``authoritative'' resolution service might be a guarantee of reliability, resiliency to failure, and atomic updates. Shorter term information is useful for short term changes in services or to avoid short lived congestion or failure problems. For example, if the actual repository of the resource is temporarily inaccessible, the resource might be made available from another repository. This short term information can be viewed as temporary refinements of the longer term information, and as such should be more easily and quickly made available, but may be less reliable.

Lastly, the publishers will be the source of much hint information that will be stored and served by the manager of the infrastructure. Since many publishers will not understand the details of the URN resolution mechanism, it must be easy and straightforward to install hint information. The publisher must be able not only to express hints, but also to verify that what is being served by the manager is correct. Furthermore, to the extent that there are security constraints on hint information, the publisher must be able to both express them and verify compliance to them easily.

2.1.2 The Client

From the perspective of the client, simplicity and usability are paramount. Of critical importance to serving clients effectively is that there be an efficient protocol through which the client can acquire hint information. Since resolving the name is only the first step on the way to getting access to a resource, the amount of time spent on it must be minimized.

Furthermore, it will be important to be able to build simple, standard interfaces to the resolution service so that both the client and applications on the client's behalf can interpret hints and subsequently make informed choices. The client, perhaps with the assistance of the application, must be able to specify preferences and priorities and then apply them. If the ordering of hints is only partial, the client may become directly involved in the choice and interpretation of them and hence they must be understandable to that client. On the other hand, in general it should be possible to configure default preferences, with individual preferences viewed as overriding any defaults.

2.1.3 The Management

Finally, we must address the usability concerns with respect to the management of the hint infrastructure itself. What we are terming ``management'' is a service that is distinct from publishing. It involves the storage and provision of hints to the clients, so that they can find published resources. It also provides security to the extent that there is a commitment for provision of such security; this is addressed below.

The management of hints must be as unobtrusive as possible. First, its infrastructure (hint storage servers and distribution protocols) should have as little impact as possible on other network activities. It must be remembered that this is an auxiliary activity and must remain in the background.

Second, in order to make hint management feasible, there will need to be a system for economic incentives and disincentives. Recovering the cost of running the system is only one reason for levying charges. The introduction of payments often has a beneficial impact on social behavior. It may be necessary to discourage certain forms of behavior that when out of control have serious negative impact on the whole community. At the same time, payment policies should encourage behavior that benefits the community as a whole. Thus, for example, a small one-time charge for authoritatively storing a hint will encourage conservative use of hints. If we assume that there is a fixed cost for managing a hint, then the broader its applicability accross the URN space, the more cost effective it is. That is, when one hint can serve for a whole collection of URNs, there will be an incentive to submit one general hint over a large number of more specific hints. Similar policies can be instituted to discourage the frequent changing of hints. In these ways and others, cost effective behavior can be encouraged.

Lastly, symmetric to issues of usability for publishers, it must also be simple for the management to configure the mapping of URNs to hints. It must be easy both to understand the configuration and to verify that configuration is correct. With respect to management, this requirement may have an impact not only on the information itself but also on how it is partitioned among network servers that collaboratively provide the management service. For example, it should be straightforward to bring up a server and verify that the data it is managing is correct. Since we are discussing a global and probably growing service, encouraging volunteer participants requires that, as with the DNS, such volunteers can feel confident about the service they are providing and its benefit to both themselves and the rest of the community.

2.2 Security and Privacy Requirements

Although much of the information we are discussing in this document might be considered ``meta-information'', there are some important security and privacy concerns that must be addressed by a service supporting that information. By first considering the sorts of attacks that are of concern, we can then focus on the security and privacy issues that are important. The reader will notice that integrity plays less of a role here than might be expected. To the extent that servers provide access control, the information they manage will have certain integrity guarantees. Beyond that we must recognize that we are dealing merely with hint information about the location of possibly interesting resources. Therefore we believe that the benefit of providing integrity guarantees beyond those provided by the servers themselves does not outweigh the cost.

Because the majority of the activity will be the distribution of hint information, the threats of concern are those affecting the maintenance of correct information to distribute and the availability of the sources of information. Although it may not be completely centralized, it is clear that hint information of the sort being discussed here will need to be concentrated in order to facilitate its discovery by potential customers. Hence the vulnerable points are the sources of the information and the distribution network among them. If one assumes that there will be principals of some sort that are responsible for the information about each URN entry in the URN resolution service, then one major threat is an attacker that masquerades as a valid principal and inserts incorrect information into the service. A second threat vector results from the fact that the service itself will be implemented by a set of servers that collaborate and share the hint information critical to their activities. By masquerading as a valid server in this pool, an attacker can both provide incorrect information to clients and provide incorrect information to other servers, which those servers will then distribute. A third threat is that if the resolution service is too centralized, service can be denied by a variety of network attacks ranging from flooding the service with queries to causing various network problems that will reduce access to the service. We can turn each of these into a security goal.

* ACCESS CONTROL ON HINTS: There needs to be an authoritative version of each hint, and it must support access limited only to those principals with the right to modify it.

* SERVER AUTHENTICITY: Servers and clients must be able to learn the identity of the servers with which they communicate. This will be a matter of degree and it is possible that there will be more trustworthy, but less accessible servers, supported by a larger cluster of less authenticatable servers that are more widely available. In the worst case, if the client receives what appears to be invalid information, the client should assume that the hint may be inaccurate and confirmation of the data should be sought from more reliable but less accessible data.

* SERVER AVAILABILITY: Broad availability will provide resistance to denial of service. It is only to the extent that the services are available that they provide any degree of trustworthiness.

Ensuring privacy for clients and publishers is in some respects essentially impossible. Fortunately, assuring a reasonable degree of privacy is possible. The privacy of clients is primarily threatened by packet sniffers and servers that log requests. A server or a packet sniffer can without much difficulty record the contents of queries as they pass by and compile the information into a relation between URNs and clients. This can be combatted in two ways: by anonymizing queries through a gateway and by encryption. The gateway solution will be the most effective protection but will involve an extra step and another potential bottleneck. The encryption solution will work to a degree, but because the queries will probably be processed by widely distributed systems the decryption key will need to be widely known, seriously diminishing the protection afforded by the encryption. The reason for this is that the client's query will probably have to be parsed by several different servers in the system, and the client does not know beforehand which ones will be involved -- hence for query encryption to work the servers must all share a single key.

On the other hand, to the degree that the search process is distributed, packet sniffing at a single point is less likely to reveal data about a specific person, and is hence less threatening to privacy. Furthermore, if clients have flexibility in terms of the specific services they choose to use, they can regularly switch services in the hopes of foiling a packet sniffer watching their usual access point.

The privacy of publishers is much easier to safeguard. Since they are trying to publish something, in some situations privacy is probably not desired. However, publishers do have information that they might like to keep private: information about who their clients are, and information about what names exist in their namespace. The information about who their clients are may be difficult to collect depending on the implementation of the resolution system. For example, if the resolution information relating to a given publisher is widely replicated, the hits to _each_ replicated copy will need to be recorded. Of course, determining if a specific client is requesting a given name can be approached from the other direction, by watching the client as we saw above.

The other privacy issue for publishers has to do with access control over URN resolution. This issue is dependent on the implementation of the publisher's authoritative URN server. URN servers can be designed to require proof of identity in order to be issued resolution information; if the client does not have permission to access the URN requested, the service denies that such a URN exists. An encrypted protocol can also be used so that both the request and the response are obscured. Encryption is possible in this case because the identity of the final recipient is known (i.e. the URN server).

2.3 Evolution

One of the lessons of the Internet that we must incorporate into the development oqf mechanisms for resolving URNs is that we must be prepared for change. Such changes may happen slowly enough to be considered evolutionary modifications of existing services or dramatically enough to be considered revolutionary. They may permeate the Internet universe bit by bit, living side by side with earlier services or they may take the Internet by storm, causing an apparent complete transformation over a short period of time. There are several directions in which we can predict the need for evolution, even at this time, prior to the deployment of any such service. At the very least, the community and the mechanisms proposed should be prepared for these.

First, we expect there to be additions and changes to the mechanisms. The community already understands that there must be a capacity for new URN schemes. A URN scheme will define URNs that meet the URN requirements document[Sollins94], but may have further constraints on the internal structure of the URN. The requirements document would allow for an overall plan in which URN schemes are free to specify parts of the URN that are left opaque in the larger picture. In fact, a URN scheme may choose to make public the algorithms for any such ``opaque'' part of the URN. For example, although it may be unnecessary to know the structure of an ISBN, the algorithm for understanding the structure of an ISBN has been made public. Other schemes may either choose not to make their algorithms public, or choose a scheme in which knowledge of the scheme does not provide any significant semantics to the user. In any case, we must be prepared for a growing number of URN schemes.

Often in conjunction with a new URN scheme, but possibly independently of any particular URN scheme, new resolution services may evolve. For example, one can imagine a specialized resolution service based on the particular structure of ISBNs that improves the efficiency of finding documents given their ISBNs. Alternatively, one can also imagine a general purpose resolution service that trades performance for generality; although it exhibits only average performance resolving ISBNs, it makes up for this weakness by understanding all existing URN schemes, so that its clients can use the same service to resolve URNs regardless of naming scheme. In this context, there will always be room for improvement of services, through improved performance, better adaptability to new URN schemes, or lower cost. In any case, new models for URN resolution will evolve and we must be prepared to allow for their participation in the overall resolution of URNs.

If we begin with one overall plan for URN resolution, into which the enhancements described above may fit, we must also be prepared for an evolution in the authentication schemes that will be considered either useful or necessary in the future. There is no single globally accepted authentication scheme, and there may never be one. Even if one does exist at some point in time, there will always be threats to it, and so we must always be prepared to move on to newer and better schemes, as the old ones become too easily spoofed or guessed.

Lastly, in terms of mechanism, although we may develop and deploy a global model supported by a global scheme, we must be prepared for that top level model to evolve. Thus, if the top level model supports an apparently centralized (from a policy standpoint) scheme for inserting and modifying authoritative information, over time we must be prepared to evolve to a different model, perhaps one that has a more distributed model of authority and authenticity. If the model has no core but rather a cascaded partial discovery of information, we may find that this becomes unmanageable with an increase in scaling. Whatever the core of the model, we must be prepared for it to evolve with changes in scaling, performance, and policy constraints such as security and cost.

In addition to the evolution of resolution mechanisms, we expect that the community will follow an evolutionary path towards the separation of semantics from identification. The URN requirements document suggested this path as well, and there has been general agreement in much of the community that such a separation is desirable. This is a problem that the public at large has generally not understood. Today we see the problem most clearly with the use of URLs for identification. When a web page moves, its URL becomes invalid. Suppose such a URL is embedded in some page, stored in long term storage. There are three possible outcomes to this scenario. One possibility is that the client is be left high and dry with some message saying that the page cannot be found. Alternatively, a ``forwarding pointer'' may be left behind, in the form of an explicit page requesting the client to click on a new URL. Although this will allow the client to find the intended page, the broken link cannot be fixed because the URL is embedded in a file outside of the client's control. A third alternative is that the target server supplies an HTTP redirect so that the new page is provided for the client automatically. In this case, the client may not even realize that the URL is no longer correct. The real problem with both of these latter two situations is that they only work as long as the forwarding pointer can be found at the old URL. Semantics, in this case location information, was embedded in the identifier, and the resolution system was designed to depend on the semantics being correct.[1] There are few cases in which we can expect semantics of any sort to remain valid for a long time, but in many cases references need to have long lifespans. Most documents are only useful while their references still function.

We expect the evolution to separation of semantics from identification to move along at least three paths. The first will be to develop temporary aliases to capture the semantics currently embedded in identifiers. This will require additional translation, but it will allow for the development of semantics-free URNs. Second, we expect locally shared or private aliases to arise, again supported by a translation mechanism and allowing for the long-term storage of global, semantics-free URNs. Such an aliasing scheme may be used to permit local aliases for named resources as well as to present these aliases to users in lieu of the URNs themselves. Lastly, we expect there may be a development of global aliases. These will be more user friendly ``names'' that would be shared on a much larger scale, and might be defined in some global registry. This may include trademarked names as well as names in extremely common use. As with the other alias systems, a facility for translation is needed. However, in this case, since the system of aliases is of global scope, the translation facility will be very slow if each time an alias is translated it needs to query a centralized or even reasonably distributed global registry. In order to achieve acceptable speeds, the translation facility will need to maintain a local cache, possibly in cooperation with other nearby alias caches. Clearly this is all postulation at present, but it is provided here to demonstrate some of the scope of evolution for which we must be prepared.

A third evolutionary requirement is even more mechanical than the others. At any point in time, the community is likely to be supporting a compromise position with respect to resolution. We will probably be operating in a situation balanced between feasibility and the ideal, perhaps with policy controls used to help stabilize the service. Ideally, the service would be providing exactly what the customers wanted and they in turn would not request more support than they need. Since we will always be in a situation in which some service provision resources will be in short supply, some form of policy controls will always be necessary. For example, suppose hint entries are being submitted in such volume that the hint servers are using up their excess capacity and need more disk space. An effective solution to this problem would be a mechanism such as a pricing policy. This pricing policy has the dual effect of both encouraging conservative use of resources and collecting revenue for the improvement and maintenance of the system. As technology changes and the balance of which resources are in short supply changes, the mechanisms and policies for controlling their use must evolve as well.

To conclude, we find that there are three broad areas in which we have requirements for URN resolution: usability, security and privacy, and evolution. Usability can be viewed from three perspectives, namely that of the publisher of a resource, of the client wishing to obtain access to the resource, and of the manager of information needed to make the published material accessible to the client. With respect to security, we find that there are requirements with respect to control of access to URN resolution information for purposes of storing and modifying it, privacy of the information, and denial of service attacks. Lastly, a URN resolution service must be prepared for several sorts of evolutionary development. At the most abstract level, the architecture itself may evolve. At a more concrete level, the evolution toward the goal of separation of semantics from identification may lead to functional changes. Lastly, with the progress of supporting technologies, we expect that there will be necessary changes in the realization of the service. Any scheme must be prepared to co-exist with revisions of itself or new approaches. With these requirements in mind, we will investigate several alternative proposals.

3 Assessment of the NAPTR Scheme

What we are calling the ``NAPTR Proposal'' is a proposal for constructing a top-level URN resolution system. The term ``NAPTR'' stands for Naming Authority PoinTeR, which is a type of database record around which this proposed system revolves. This proposal represents an approximate consensus of the people working on URN resolution in the IETF at the time the URI Working Group was dissolved. As of the writing of this document, we have not seen a full written description of the NAPTR proposal, so the following description is based on conversations and email (in addition to some pseudo-code) with various members of the NAPTR group. This description represents the best of our ability to understand that proposal, but we make no guarantees of its correctness.

3.1 Brief Explanation

A URN has been defined to be of the format:

```
        URN:[NSI]:[OS]
```

where NSI is a valid NameSpace Identifier and OS is an Opaque String which is interpretable only within the context defined by the NSI. The proposed model is that NSIs are allocated by some central authority to identify naming schemes. Within each naming scheme there would be many naming authorities that are charged with distributing globally unique names from within their individual namespaces. Each naming scheme would also have a global plan for the delegation of naming authorities and the namespaces they manage.

Given that a client has a URN that needs to be resolved, the resolution process proceeds as follows:

1. The client pulls the NSI token out of the URN, and issues a DNS lookup on ``[NSI].urn.net''. This query returns an NAPTR record or an SRV

record. The intention of an NAPTR record is that it tells how to find a Naming Authority and lists the services offered there. In order to determine the Naming Authority for a given URN, an NAPTR contains a rewrite rule which is used to rewrite the URN into a string identifying the naming authority. An SRV record describes an authoritative URN resolution server and supplies the necessary information to contact it.

2. If the client received an SRV record, it connects to the service described on the appropriate port and attempts to resolve the URN. If this attempt is successful, then the resolution is complete.

3. If the client received an NAPTR record, it applies the enclosed rewrite rule to the URN. This results in a string that is issued in another DNS lookup. This lookup returns one or more NAPTR records and SRV records, which are processed with steps (2) and (3). There is an algorithm for preventing looping and for selecting which rewrites to try first.

3.2 Definition of the New DNS Records

These are the structures involved, from the Perl code implementing the client side of this system, as of April 1, 1996:

```
# This structure represents one of the NAPTR records returned fron the registry
$naptr_rec = {
        services =>     $string,    # List of resolution services offered?
        pattern  =>     $string,    # regexp substitution pattern
    };
# This structure represents one of the SRV records returned from DNS
$srv_rec = {
        priority =>     $string,    # preference
        weight   =>     $int,       # weight if preference is equal
        port     =>     $int,       # protocol port
        target   =>     $string     # host to contact
    };
# This structure represents the parsed values in the Service field of the
# NAPTR record.
$service_rec = {
        flag            => $string,    # flag if not a real value
                                       # 0 = I don't know, keep going
                                       # 1 = I know all
                                       # 2 = normal case
        protocol        => $string,    # network protocol
        ServiceClass    => [ @list ]   # list of service classes
    };
# This is a builtin list that contains the valid Services that this
# client speaks. The API to the potential library should have some way
# for the client to add/remove things from this list.
%valid_ServiceClasses= ("n2l"   => 1,  # URN to URL
                        "n2ls"  => 1,  # URN to URLs
                        "n2r"   => 1,  # URN to resolver
                        "n2rs"  => 1,  # URN to resolvers
                        "n2c"   => 1   # URN to URC
        );
```

3.3 Assessment

The NAPTR scheme has the very useful property of being implementable right away using existing infrastructure, i.e. the DNS. This ease of implementation and deployment makes it imperative that there be an evolutionary path making possible the satisfaction of any requirements that are not completely satisfied. We will proceed through the various requirements, assessing the extent to which the requirements are satisfied, and examining key areas in which the requirements are either not satisfied or not addressed by the NAPTR specification.

3.3.1 Assessing Usability

From the client's perspective, the NAPTR system seems to meet most of the requirements. Since its protocol is based on DNS lookups, it is well known and already widely implemented. Building a user interface for the client side is not too difficult, and essentially involves the question: how does the user control the parts of the resolution process that involve decisions? For example, the NAPTR system allows the client to select sites that return certain types of information, for example, whole resources versus only their URCs, and this control could be passed to the client. Another issue surrounds the fact that the result of the resolution will likely be a collection of SRV records; perhaps some heuristics in the browser might select the order in which to try these records, taking into account locality or other specialized information. In some cases it may be desirable that the client make this decision.

However, when we consider performance, it is hard to tell how fast the system will operate. In part, the reason is that the specification of the system is not clear about how the system will be organized in typical situations, despite that fact that performance will be tightly coupled to the structure of the namespaces. Certainly when the service is initially deployed the performance ought to be pretty good regardless of structure. The topmost level of dispatch (i.e. the registry of NSIs) can be cached locally for long periods of time, so at first not more than one or two DNS lookups should be necessary to locate an authoritative service.

3.3.1.1 Long Term Effects of Rewrite Rules

The effect of the rewrite rules in the NAPTR system is not easily characterized. In general systems based on rewrite rules tend to be difficult to configure and are not generally understood by ``mere mortals''. There is no question that the rules are flexible enough to accomplish the tasks required by a top-level URN resolution system. On the other hand, the desire of the publisher and user to maintain long-lived URNs may be compromised by the quirkiness and difficulty of managing rewrite rule databases.

Initially, we expect that most of the URNs within a namespace will be found using a simple, unified set of rules. However, as time goes on we can expect the namespace to become fragmented to an increasing degree, as sets of long-lived URNs are served by different authoritative services. Fragmentation will have a direct impact on performance of this system, because it will mean sending additional records back to the client and in many cases will also require additional DNS lookups. We expect that these performance problems will not effect the users of the system because, rather than suffer performance problems, the administrators of the system will simply discourage the sorts of activities that would cause performance problems. While this may solve the performance issues, it has the net effect of more tightly binding naming authority to name resolution and of restricting the publisher's mobility.

This inability to gracefully respond to fragmentation is the result of the way the rewrite rules work. The rewrite rules are a very flexible technique for _generalizing_ classes of URNs; however in this system all _discrimination_ of URNs is done on the client side. For example, consider the URN ``URN:INET:mit.edu/lcs/ana/mesh/paper.ps''. Suppose that MIT's URN resolution system is set up so that each lab runs its own server; hence LCS would run one to cover its collection of URNs. Then the resolution process would select out ``mit.edu'' as the naming authority, and do the DNS lookup ``mit.edu.urn.net''. This would return another record with a rewrite rule that selects out the lab name, in this case ``lcs'', and perhaps looks up ``lcs.mit.edu.urn.net''. This returns an SRV record indicating the authoritative URN server for LCS. (Keep in mind that this is only one way of setting up these rules!)

Now suppose that the ANA group decides to run an experimental new kind of server. There are two ways to cause the top-level system to point to this new server, and both of these methods result in an incremental cut in performance. First, a new record could be added to the system so that when ``lcs.mit.edu.urn.net'' is looked up, two records are returned, one SRV record that points to the authoritative LCS server, and a new NAPTR record that rewrites ANA URNs to ``ana.lcs.mit.edu.urn.net''. When that is looked up in the DNS, a SRV pointing to the experimental service is returned. This is the process of installing an _exception_ into the top level resolution mechanism (i.e. all LCS URNs _except_ the ANA URNs go to server A, while the ANA URNs go to server B.)

While having one exception is not necessarily a problem, if any number of exceptions are installed, a large number of records will be sent back in response to the DNS lookup. In order to get around this problem, the second method must be used. The second method pushes the generalization back a level, and changes the rewrite rule at the ``mit.edu'' level to select out the whole ``lcs/ana'' token rather than simply the ``lcs'' token. This is not always possible, depending on how the namespace is set up; in some cases two separate rewrite rules will be required at the ``mit.edu'' level, one covering labs that do not have sub-departments and one covering the labs that do. In order to make the system work, each department under LCS might need a separate SRV record, many redundantly pointing to the generic LCS server. Then when the ``lcs/ana'' token is matched, the URN is rewritten into ``ana.lcs.mit.edu.urn.net'', and the correct server is located with a single additional DNS lookup.

In summary, the cost of exceptions can be borne in two ways: first, in additional records passed back, additional client-side computation, and additional DNS lookups; or second, in a considerable expansion of the size of the DNS database for that particular part of the namespace.

3.3.1.2 Usability for Clients

From the perspective of a client, the system should work well. The client side of the protocol is fairly easy to implement, the only difficulties being loop avoidance and heuristics for choosing which servers to try first. The performance of the system could become poor, but as we speculated above it is more likely that other positive attributes will be sacrificed prior to performance.

One way that the system could be improved from the client's perspective would be to implement some security features. For example, the resolution system could set aside room for authentication information in the SRV and NAPTR records. The client could then verify the authenticity of the records using whatever security infrastructure is available (the authentication protocol itself should probably remain separate from the resolution system).

3.3.1.3 Usability for Publishers

From the perspective of a publisher, many important issues are left unspecified by the NAPTR system, leaving it up to each specific naming authority to select or invent a policy.

* Publishers who want to use a different URN resolution service will need to communicate this somehow to the naming authority who gave them the names, so that they can modify the NAPTR and SRV records in their registry. Nothing has been specified explaining how this is done or how easy it is. In theory entities can run their own naming authority registry but that has the net effect of slowing down the system.

* Publishers who acquire names from a given naming authority are forced to contract with that naming authority to ensure that the naming authority continues to support the publisher's namespace in their registry. This must be done for as long as the names remain valid; if for some reason the naming authority ceases to function, someone else must take over that service. Except for the owners of the top-level namespaces (i.e. directly located from the first rewrite rule, and not delegated), this scenario is not guaranteed to foster the maintenance of long-lived names.

* Publishers do not in general have direct control of the top-level resolution of their names, and, like the DNS, there is no authentication model for validating information obtained from the system. Since the system is intended to be global, it stands to reason that there will be no way for the publisher to maintain tight control over the process of resolution. However, with the right design, it is possible to make it very difficult to misdirect clients with forged hint information, by implementing an authentication protocol. Unfortunately, a facility for authentication is not built into the NAPTR system and will not be easy to add.

* Publishers do not have any means at their disposal to make temporary changes to the resolution process. In order to make any kind of change, that change must be registered with their parent Name Authority, a process which may take time, may be subject to a time-consuming review to ensure that it is a legal change (i.e. not a security violation), and may cost money. All of these potential hassles are undesirable when a change must only be made for a brief time. The NAPTR system does not specify how temporary changes would be handled, instead leaving it at the discretion of the NA Registry management to define a policy.

3.3.1.4 Usability for Naming Authorities

From the perspective of a naming authority, exceptions will be expensive for many reasons. First, it means an increase in the size of the registry. Second, and more important, it means an increase in the complexity of the registry. It will be very important for naming authorities to understand the tangle of rewrite rules that make up their registry. It is likely that naming authorities will need to set down very specific guidelines delineating a rewrite rule policy and security policy in order to keep the registry working. But the structure of these policies is neither specified nor suggested by the NAPTR system as we understand it.

It is unclear whether rewrite rules provide increased flexibility or merely an increased need for security. Given a system of arbitrary rewrite rules, it is difficult to determine the layout of the namespace, i.e. who has authority to specify which rewrite rules, unless the rules are constrained in some systematic way. These constraints may make the rules no more flexible than a simpler system -- but this is difficult to assess because the necessary constraints have not been suggested.

Many naming authorities may proceed on the assumption that exceptions and other things that make their lives more complex are simply not that important. However, easy implementation of exceptions is vital to the maintenance of long lived URNs (by long lived we mean perhaps ten or a hundred years). There are a number of ways in which exceptions are likely to arise: naming authorities split up into pieces, naming authorities go out of business and are split up, corporations merge and split, students at universities take collections of URNs with them, etc. It is hard to predict the sorts of uses to which these systems will be put, but fragmentation over the course of time is a given. In the NAPTR system, the installation of exceptions is possible but it will be difficult enough that in many cases URNs will be thrown away rather than bear the cost and hassle of maintaining them. This will have the singular effect of negating their distinctive feature of being immutable over the lifetime of the named resource.

3.3.2 Assessing Security and Privacy

Like many systems designed for the Internet, issues of security have not been given a full treatment in the design of the NAPTR system. Part of the difficulty stems from the fact that it is built on top of the DNS, which is well known to lack ``security considerations''.

3.3.2.1 Security

Because the NAPTR system is designed around the DNS, issues of security have not been addressed very thoroughly. The systems of rewrite rules involved with the NAPTR system also have the side effect of muddying the waters surrounding security. In part the reason for this is that the security restrictions are defined on an individual name authority basis, and hence do not imply a global security policy.

The NAPTR system does not clearly and securely specify delegation of naming authority, except at the uppermost level of the NSI registry; presumably some authoritative entity regulates new entries to the NSI registry. At all other levels collections of rewrite rules crafted by individual name authorities specify delegation of authority, if any. In some sense authority is never actually delegated, because any name authority higher up in the resolution path has the power to revoke authority to those sub-authorities below them by inserting or deleting the appropriate rewrite rules. Furthermore, the specification of delegation is at best difficult to understand because the systems of rewrite rules can be arbitrarily complex. In summary, (1) the ``higher level'' name authorities have too much power (which evokes the question of whether hierarchies of naming authority are likely to arise at all), and (2) the rewrite rules make understanding the delegation of name authority difficult for the _operators_ of the name authority registry, as well as the publisher.

3.3.2.2 Privacy

In general, the NAPTR system does a good job of providing privacy to publishers. The information on URN resolution servers is entirely private and need not be known at any of the higher levels in the resolution process. Name allocation by publishers within their designated block of namespace is unconstrained and private. The data stored outside of the publisher's physical control is kept to a minimum, and represents a minimal invasion of privacy.

The privacy of clients of NAPTR is reasonably well protected. In general, all of the network traffic associated with the client will be the result of DNS lookups, so a traffic analysis performed at a point outside the client's LAN will not be able to determine which principal on the client's LAN is performing DNS lookups. However, there is no way to foil a traffic analysis performed on the LAN itself.

3.3.2.3 Resistance to Attack

Because it is based on the DNS, the NAPTR system inherits its susceptibility to many kinds of attack. Falsified information can be provided by fake servers, and no authentication model has been proposed. Furthermore, incorrect information can be provided by real servers that have been corrupted. Provision of falsified information can be done at any level of the name authority hierarchy, potentially denying service to both publishers and clients.

Denial of service is one issue that is not addressed effectively by the NAPTR system. Resolution pathways alternative to those defined by the rewrite rules do not really exist. The NAPTR system implements a model of distributed authority which is intended to make it difficult to corrupt the system from above. That is, the power to make alterations to a NA Registry is alone held by the maintainer of the registry. However, in the event that a registry is corrupted, the publisher whose data has been corrupted has no recourse or means to repair the damage outside of complaining to the maintainer of the corrupted NA registry. If the maintainer caused the problem deliberately, this may not be effective.

Finally, packet sniffing can be used to make records of queries, although this is true of a wide variety of protocols.

3.3.3 Assessing Evolutionary Requirements

The most critical requirements for a URN resolution system are those that make it possible to update and upgrade the system in the future. It is unreasonable to expect the first implementation of a URN resolution system to have all of the features that will eventually be desired; in many cases the need for such features has not yet arisen. Soon after a URN resolution system is accepted, a large base of compatible client software will be developed. After this initial implementation effort, it will be much more difficult to update the client software to understand new protocols, because the need is much less immediate. If the capacity to evolve is not built into the initial client protocol, future systems will be much harder to deploy.

The requirements for URN resolution systems focus on three key areas in which evolvability is important: ease of deployment, evolution toward the separation of semantics from naming, and extensibility at a variety of levels. The NAPTR system's strong suit is ease of deployment; it uses the DNS as the backbone of its distributed registry and requires only modifications to the DNS software.

3.3.3.1 Ease of Deployment

There is no question that the NAPTR system can be deployed quickly.

* The client-side protocol uses a modified version of existing DNS software to locate the proper URN resolution server.

* The construction of Name Authority registries requires a modified version of existing DNS server software. Initially, the registries will be extremely simple, requiring the addition of only a few new records to existing DNS server databases.

* The new records will be located in their own portion of the DNS namespace so they need not interfere with the existing DNS hierarchy (although it is not required that they reside in the ``*.urn.net'' domain -- this depends on the rewrite rule configuration).

With the pressures of cost recovery, commercialization, and social management, economic mechanisms are coming to the Internet. While no economic model is specifically suggested, it is easy to see how a tenable economic model would arise in the context of the NAPTR system. Given the NAPTR system, it seems likely that Name Authorities would hand out blocks of their namespaces and assess incremental charges per unit time during which they keep the registry information for each block of namespace valid. Additional charges might be levied in the event of complicated exceptions, etc. This model seems likely because it parallels the incremental cost of maintaining the DNS servers and registry databases. This model also makes being a Name Authority a lucrative business, since the publisher whose URNs reside there must contract with that specific Authority or else change their URNs.

Given such a model, the naming authority market would be relatively monopolistic. Although it would encourage rapid deployment, at the same time it would tend to discourage long-lived URNs. In contrast, the market in URN resolution servers should be a highly competitive one under the NAPTR system.

3.3.3..2 Naming Without Semantics

In order to meet the long term goal of long-lived URNs, it is important that semantics be separated from naming to as great a degree as possible. This is not a new concept; it is interesting to note that the original design of the DNS called for three distinct layers of naming: a routing-conscious layer (i.e. the IP address space), a unique long-lived naming layer (i.e the current DNS namespace), and a layer of user friendly aliases which was never implemented.

The NAPTR system as specified neither encourages nor discourages semantics-laden names. We do not suggest that restrictions should be placed on URNs -- certainly publishers must be allowed to choose whatever names they want -- but providing support for user friendly names that cannot be used as embedded references would at least encourage the separation of semantics from naming.

The issues surrounding aliases are considerable, and will not be solved right away. There are plans to include support for user friendly names in the NAPTR system, but at present it is unclear exactly what support will be provided.

3.3.3.3 Extensibility

Several kinds of extensibility are required of a URN resolution system.

* A URN resolution system must be able to support future naming schemes. The NAPTR system exhibits a great deal of flexibility in supporting new naming schemes through the use of arbitrary regular expression based rewrite rules.

* A URN resolution system must be able to support new authentication protocols. The NAPTR system does not specify any form of authentication. In order to add authentication protocols later, either the client support would need to be modified or the protocols would have to be implemented as a separate system, with distinct client support. In either case, modifications to the client will be required.

* A URN resolution system must be able to support new URN resolution services. The NAPTR system provides this extensibility using the SRV records, which specify host, port, and protocol. The SRV records also provide information about the types of services provided by the specified service. This design is highly extensible to new services.

* A URN resolution system must include a gateway protocol that allows other ``top level'' URN resolution systems to be used. Without such a feature designed into a client-side implementation, it will be very difficult to upgrade to a system that offers better features. It will also make it impossible to allow clients to choose from multiple competing top-level systems. As currently specified, the NAPTR system does not require clients to support a gateway protocol. Without such a protocol designed-in it will be extremely difficult to phase in new URN resolution systems. In part the reason for this is that the client side of the NAPTR system exclusively uses DNS queries to locate resolution information. In order to cause a NAPTR client to proxy to another system, the DNS would need to be egregiously hacked. Fixing this problem is simple; it should be specified that clients implement, in addition to the normal NAPTR client protocol, a very simple gateway protocol which sends the URN to a proxy and gets back a list of hints, perhaps in the form of SRV records.

In summary, the NAPTR system is extensible in a many of the required modes, but presently lacks the most important extensibility, a protocol for using another top level system through a gateway. It also fails to leave room for authentication protocols which may become desirable. This problem could easily be corrected; for example, SRV and NAPTR records could contain a field for authentication information.

4 An Alternate Model

Not all the requirements outlined in this draft are of immediate concern. However, many of them will become more important as the extent and importance of URN resolution grows. Since URNs derive their utility from being long-lived, it stands to reason that one or more usable URN systems will be needed for as long as the URNs are used, and that entirely new systems may evolve as well. In this section we present a different URN resolution model as an example of a system that provides more of the flexibility specified in the requirements. In section 5 we assess exactly what kinds of extensibility are needed to implement such a system.

4.1 High Level System Architecture

In this section we lay out a high level architecture that is open and extensible. It is based on five types of entities, connected to each other by flexible gateway protocols. New developments in any of these parts should entail a fairly smooth transition.

* Authoritative URN resolution servers. These servers are operated by or on the behalf of publishers. They provide authoritative resolution of specific collections of URNs. These servers are the endpoints of the system we are describing here. The implementation and protocols involved are not discussed in this document.

* Name authority registries. These are servers that maintain the authoritative map of ownership over a given root namespace. For each owned namespace, authoritative URN resolution servers can be specified to handle the URNs in that namespace. There would likely be separate registries for ISBN, INET, and ISO.

* Distributed client interface systems. These are distributed systems that serve the information stored in name authority registries, specifically hints for finding appropriate authoritative URN resolution servers. Through distribution and replication they provide an efficient interface through which clients can locate the right authoritative URN server. Hints may be injected by clients into these distributed systems as well, for many reasons which will be explained later. Hints are returned to the client along with the official hints from the registry. When a distributed client interface receives a URN that falls outside of the namespace it serves, it may be a good idea to require that system to pass the URN on to a system that does serve that namespace. This way the client doesn't need to worry about choosing a distribution system based on the URN.

* Top Level ``URN:*'' Registry. In order to maintain order in the ``URN:*'' namespace, there will need to be a registry that allocates non-conflicting naming schemes. This registry would also identify for each naming scheme the official name authority registry for that scheme, as well as a list of distributed client interfaces that serve that scheme.

* Clients. These are users with URN-enabled applications, such as browsers. The application sends a query containing the URN to a local distributed client interface, and eventually a collection of hint information is returned. This hint information typically points to URN resolution servers which may resolve the URN in question.

The system we are trying to describe here has primarily to do with the name registries and the distributed client interfaces; the other elements, that is, the clients and the Authoritative URN resolution services, are more on the fringe of the system. In the next few sections we will go into more detail about how the name registries and client interfaces might work.

In section 4.2, we specify a new architecture for a global namespace. This architecture builds on the decisions already reached in the URN community, and presents an expanded architecture for delineating naming authority that maintains compatibility with existing ``legacy'' namespaces while adding expressiveness and power. The name authority registries will implement this architecture.

In section 4.3, we sketch out the workings of one possible implementation of a distributed client interface system. Here we discuss how hints are intended to work, why they are a good idea, and what additional protocols are necessary to implement them.

In the remainder of this section, we will lay out the overall architecture of this alternate model. The effect we want to achieve is a decoupling of the keeping of registries from the production and maintenance of distributed client interfaces. This way, new types of distributed system can be installed more easily, and new registries (i.e. new namespaces) can be added more easily. Toward this end several protocols need to be implemented between the parts of the system.

4.1.1 Clients

In this system, clients typically send their queries to the distributed system specified in their application's configuration. This configuration may specify a backup system in the event that the usual system is down or sluggish. The queries contain URNs to resolve, and may also specify various information about the perferred type or format of the data returned. If the system they use does not serve the URN they want directly, the system is obliged to proxy the query to a system that does. Information about which systems serve which naming schemes is available from the central ``URN:*'' registry. When the hints are returned to the client, the client may authenticate them to make sure they are genuine. The hints are then used to continue the resolution process by looking up the specified authoritative URN resolution servers and querying them about the URN in question.

It is important to note here that we are assuming that all of these systems are used fairly infrequently. They form the last resort -- after all other hint information has failed. We assume that servers of documents would generally maintain collections of hints to supply with each document. These hints would as a rule be sufficient to locate the document; if the hint telling the location is out of date, the hint telling the most recent authoritative resolver might not be. However, when a URN is discovered but all its associated hints are outdated, the systems we are discussing here are necessary to make sense of it.

When more up-to-date hint information comes back to the client, the client must then use that information to complete the resolution process. This typically involves contacting the suggested authoritative URN resolution services, using whatever protocol is convenient. As a rule, the ``official'' hint information will be sufficient. An authenticator included with each hint can be used to verify the identity of the hint's author. In the event that the official information fails or is for some reason suspect, the unofficial hint information can be used. In some cases, there will be unofficial hints that are authentically written by the owner of the namespace or the author of a document; these would be most likely to be correct. The authentication can proceed via whatever infrastructure is convenient. At the moment authentication is not in wide use, but hints can easily be specified to have a signature or certificate field.

4.1.2 Distributed Client Interfaces

In this system, distributed client interfaces maintain their internal organization through their private protocols, and apart from that maintain three gateway interfaces with the rest of the world.

* Standard Client Interface. This does not rule out the possibility that a distributed system would understand a specialized client interface as well (with extra features, etc.) It merely specifies that all distributed client systems should understand the same minimal gateway protocol, for example, sending a URN and getting back a collection of hints. Such a protocol makes proxying possible.

* Hint Passing Interface. This interface is necessary so that hints injected into one distributed system migrate to the others. When a submitted hint is found to lie within a given naming scheme, that hint is forwarded to all other distributed client systems that also serve that naming scheme. The information about which other systems serve a given naming scheme is retrieved from the ``URN:*'' registry.

* Bulk Transfer Interface. This interface makes it possible to receive a name registry's database and subsequent updates in bulk. The data can optionally be translated to have a particular direction of hierarchy so that naming schemes with incompatible hierarchy schemes can coexist.

4.1.3 Name Registries

In this system, the name registries support the other half of the bulk transfer interface. Some registries may want to select the systems that serve their databases. Because transferring the database involves a reasonable amount of bandwidth, it will probably be necessary to avoid sending it to impostors, etc. The database itself has a value as information as well, so it may even be worth while to encrypt the bulk transfers and updates.

The name registry for ``URN:*'' is special; it is the meta-registry. There is exactly one of these and it is analogous to the NSI registry in the NAPTR proposal. The main difference is that the client never accesses it unless it is necessary to locate a distributed client interface to use. In general, its main purpose is to store for each registry information about which systems are serving that database. The registry also contains contact information for the systems and registries and the ports on which they support the various standard gateway interfaces.

4.1.4 Other Parts

There are other pieces of infrastructure that we don't address here. First, we don't specify very much about how interfaces to Authoritative URN services work. In part the reason for this is that there are many unresolved questions about how to handle meta-data, versioning, format selection, content negotiation, etc., which are all essentially irrelevant to the top level of URN resolution.

Second, we don't specify how authentication works. We are designing room for authentication protocols into the system, but we leave what the security infrastructure does with the authentication certificates open. A workable security infrastructure would be highly useful, and hopefully one will be developed.

4.2 The Architecture of a Global Namespace

Our alternative system centers around a more sophisticated model for designating naming authority. Currently, models of name delegation are implemented and managed by neutral parties and standards organizations. For example, delegation in the DNS namespace is administered by IANA, ISO administers the ISO namespace, and ISBN is administered by an association of publishers. While each namespace has its own set of rules and policies, the data they maintain is structurally very similar.

Unfortunately, all three of these current models have shortcomings. The DNS is the namespace most integrated into popular computer culture, but this sudden success has begun to undermine its elegant distribution model. The ISO naming scheme is flexible, powerful, and has a good distribution model, but is underutilized, in part because the names are formed from numbers and do not have a simple mnemonic form. The ISBN namespace is very effective at naming books but is rarely used outside that domain.

4.2.1 Marketing vs. Distribution Models

There is a need for an effective model of name delegation, and that effectiveness must be maintained in the context of market forces. The DNS is a good example of a well thought-out system that is currently succumbing to the desires of an increasingly commercial Internet. Because its distribution model is dependent on subdelegation and the formation of hierarchies, as the root of the DNS namespace grows disproportionately to the levels beneath it an increasing fraction of the namespace gets replicated monolithically rather than being distributed.

Unfortunately, subdelegation turns out to be an unpopular activity in the ``*.com'' domain, for example, resulting in a very flat namespace, for several reasons:

* The desire for semantically significant names, the need for complete control over the ``look and feel'' of the names (that is, except for the ``.com'' part...) and the threat of competitors stealing them

* Lack of any obvious organizational hierarchical structure

* Reluctance to share a common root (other than ``.com'') with a competitor, thus becoming dependent on a DNS server over which they share authority

Given these reasons for the failure of the DNS to adapt to changing situations, how can a new system be built that avoids these pitfalls? One key to answering this question is in the recognition that the present demand for semantic nicknames is not going to disappear. Furthermore, as long as names have a mnemonic or semantic component it will be impossible to expect the namespace to have any particularly convenient structure. We therefore must resign ourselves to keeping a big flat namespace which must be broken up into convenient-sized pieces, distributed and replicated.

It is important to note that this assumption does not belittle the importance of semantics-free namespaces! On the contrary, semantics-free namespaces are important to the goal of having long-lived names. However, for a semantics-free namespace to be popular there must also be a semantics-laden layer of aliases or nicknames which serves as an ephemeral and dynamic human interface to the semantics-free namespace. These nicknames would never be included in machine-processable references; whenever a link or bookmark is kept, the semantics-free permanent name is substituted for the nickname.

In conclusion, we need to bite the bullet and produce a distributed client interface system capable of satisfying lookup queries against a large catalog of names that does not necessarily exhibit a dependable hierarchical structure. Such a system must be both distributed and replicated, preferably automatically. To the extent that the system is based on a large centrally maintained master list there is a need for alternate maintenance processes in order to guarantee that the system would operate despite failures of the centralized portion of the system. Such a system is more complex than the DNS, but, especially in the light of recent improvements in cross-platform compatibility, it is within the realm of possibility.

4.2.2 Conceptual Extension of the Root Namespace Registry

We are also suggesting an extension and standardization of the current notion of ``root'' namespace delegation. Rather than have central registries that record only a few top level naming authorities and expect those authorities to support sub-delegation with their own resources, registries compliant with this extended model would be designed not only to accommodate a large number of delegations at the top level, but also to provide means of ``officially'' registering sub-delegations that fit into the standard delegation model.

That is, when a naming authority _officially_ delegates a portion of its namepace to another entity, that entity must register authority over that portion with the central registry. Note that official delegation is never required, but the option to do so always exists. For example, suppose MIT owns the ``URN:INET:edu.mit/*'' namespace, which in turn contains the namespaces of its various labs and departments, including ``URN:INET:edu.mit/lcs/*'' and ``URN:INET:edu.mit/eecs/*''. MIT has the option either to delegate those parts of their namespace to the labs that use them officially, or to formulate its own internal security policy to determine who has permission to alter what portions of their URN resolution information. A system set up this way has a number of advantages:

1. It makes it possible for existing portions of a naming authority's namespace to be transferred to a new owner with no strings attached. When ownership is transferred officially, the original owner is no longer in control of any part of the resolution process for that space of URNs; in return the original owner is absolved from all responsibility over that space. When naming authority is delegated in this model, queries are referred to the current owner of a namespace directly from the top level rather than being proxied by the previous owner, as would be the case with the DNS.

2. Naming authority is clearly and securely delineated. The mapping of

URNs to URN resolvers is performed in a two step search, first determining which entity owns a given URN, and second using whatever information is provided by that entity to locate a resolution service. This guarantees that resolution information provided by an official naming authority cannot interfere with URNs owned by other official naming authorities.

3. Registering namespaces ``offically'' with the root namespace registry has the effect of flattening the namespace. However, as we have argued previously, semantics in naming will tend to keep namespaces relatively flat anyway. We expect that the additional entries introduced when the namespace is flattened in this way will not increase the size of the root namespace by a significant factor.

Standing alone, this model is insufficient. There are two major objections to it which must be addressed. First, there is an implicit assumption that name authority is to be designated on a certain ``shape'' of boundary. This shape is analogous to a hierarchical file system, where lexical prefixes such as ``URN:INET:*'' are analogues of subdirectories. Write permission on a directory and write permission on the files contained within it can be the property of different owners. That is, IANA might own the subspace ``URN:INET:*'' but would delegate authority over subspaces such as ``URN:INET:edu.mit/*'' to other entities. Most legacy namespaces seem to be easily transformed into this form (for example, the DNS names we have used have had their domain names reversed.) If some new namespace arises which uses an incompatible model for delegation of authority, a gateway protocol can be employed to resolve those URNs.

The second objection is not so easily answered. This objection has to do with the potential danger of having a large centralized registry. Even if distributed systems can be built to serve it efficiently, there remains the danger that the centralized portion breaks or becomes corrupt. We believe that these problems are mitigated by the hint system described later in this document. Essentially, if the centralized registry breaks, the distributed client interface systems will continue to serve the most recently obtained data. By injecting authenticatable hints into the distributed systems, incorrect data can be corrected and missing data can be supplied. Since the system allows for impromptu and easy hint correction and update, incorrect behavior at an central service can be mitigated, as will be seen below.

4.2.3 Economic and Political Implications of Registries

In order to ensure consistency, some authority will need to be in charge of maintaining each registry. The implementation of a distributed client interface can be entirely decoupled from the maintenance of the registry itself, and can furthermore be done by multiple competing services. The choice of which top-level system to use would be offered to the user, making new technology and competing services available to existing clients. Different namespaces (i.e. DNS vs. ISBN) might have separate official registries, but in general the distributed client interface systems would either distribute all the data from each of the registries or proxy the queries that are not handled to a different distributed system that does.

The authority that keeps the registry is not required to be the authority which maintains the root of that namespace. For example, in the Internet today IANA manages the root DNS namespace (that is, acts as a naming authority), while the NIC maintains the root DNS servers (that is, acts as a client interface). Today, in order to allocate a new domain name, an applicant must pay $50 per year to IANA, which then enters that new domain into its databases. Further names within that sub-namespace are free, but the owner of the namespace is responsible for maintaining the servers that resolve them. In this new system, the owner of the root namespace, (suppose it is IANA) can sub-delegate subspaces of the root to other entities. Those entities must then register the subspaces with the ``URN:INET'' registry, possibly paying a registration fee at that time. Once a sub-namespace is registered, any names falling in that subspace are resolved using the information supplied by the owner of the namespace; so registration is only necessary when a namespace is delegated to a different owner, or when an owner wants to change their information in the registry. The creation of new names in a registered namespace is not regulated in any way by the registry.

To make it simpler to construct new distributed systems and to add registries for new namespaces, it makes sense to define a standard protocol between registries and distributed systems. This protocol would allow bulk access to the database, returning the data in a canonical format. It will also provide updates to subscribers using the same canonical format. This canonical format would present hierarchy in a standardized way by making all the trees branch in a uniform direction. Once the hierarchies are all branching the same way, the entries in the database can simply be listed in lexical order, and the process of breaking it up for distribution can begin. For example, DNS names would need to be rearranged so that the hostname tokens are reversed, i.e. ``lcs.mit.edu'' -> ``edu.lcs.mit''. Similar techniques can be found that allow most legacy namespaces to fit this canonical form. Many namespaces, such as ISO, need no transformation since they are already hierarchicalized in that order. Many distributed systems might choose to maintain their own replicated copy of those registries that they distribute and serve, in order to figure out how to best organize the distribution.

4.3 Architecting Distributed Client Interfaces

As we have indicated in the previous sections, in our model we assume an essentially flat namespace. It may be the case that parts of the namespaces we serve do exhibit a useful form of hierarchy; however since we cannot depend on this we assume a worst case. Given this assumption, in order to provide scalable access to the naming system, we must build a system of cooperating name servers which distribute and replicate the database. We call systems providing this function (i.e. an interface between the client and the name authority registries) distributed client interface systems.

There may be many distributed client interfaces to the same namespace. This will pave the way for new client interfaces and new distribution mechanisms to be put into place. Each distributed client interface system acquires the registry databases in bulk using the aforementioned bulk transfer interface. It is advised that all client interface systems provide access to the entire space of URNs; for those spaces in which authoritative information is not served directly, resolution requests should be proxied to another service.

While building these systems is not trivial, recent improvements in cross-platform compatibility mean that the largest impediment to constructing such a system in the near future will be the existing infrastructure, should it be too inflexible to accommodate a new resolution mechanism. We have been working out some of the details of a distributed client interface, and hope to have a testbed implementation in the next few months. The next section sketches out our basic strategy for distribution and replication.

4.3.1 Distributing and Replicating a Flat Namespace

Our distribution system relies on the fact that the database to be distributed can be collected in one place temporarily.[2] Once the database is assembled, it is broken up into pieces of approximately equal size. These ``chunks'' of the database must be of a reasonable size, since each participating server will be responsible for storing and serving one or more ``chunks'', as well as for having a certain amount of excess capacity to store updates and hints. Once the database is broken up, new nodes are constructed to form an n-ary search tree. In order to make the nodes more uniformly sized, ``leaf elements'' may be shifted upwards into the nodes forming the search tree.

Once the search tree is planned out, it is installed on a set of servers in the system. Once it is installed and running, clients begin to use it. As the installed servers become loaded, those parts of the tree experiencing the heaviest loads are replicated on other available servers. This process of replication results in the ``growth'' of a robust distributed system from a planted seedling. Actually getting this growth process to work requires careful analysis and simulation, as well as the discovery and implementation of a working set of protocols. However, there are benefits:

* If a major rebalancing is needed, a new system can be ``grown''. This means that the update procedures can be made simpler (i.e. no need to redistribute the data in the field).

* The system as a whole must be robust. In some sense, it is always in a state of dynamic balance, some parts trying to grow while others shrink. Perturbations on the scale of individual servers are part of the day to day process.

* All configuration should be as automated as possible. Furthermore, in terms of security, in a highly dynamic and self-configuring system composed of relatively untrusted nodes, the dynamicism counteracts many of the security weaknesses brought about by the fact that the cooperating servers are untrusted. The reason for this is that it is impossible to predict which portion of the database a given server will be serving, thus reducing the likelihood of gaining advantage from serving a particular portion.

Replication is handled using a two-tiered structure. Groups of nearby replicated servers form _replication_groups_. A leader is chosen from each replication group to join a _supergroup_ which represents the totality of the system responsible for serving one particular part of the database. Updates to the database are flooded to the members of each replication group in a way similar to floodd[Danzig]. The administration of the system is accomplished by the leaders of the groups and supergroups. The agent responsible for generating the system in the first place (i.e. collecting the data, etc.) would also maintain the supergroup leaders. Note that the system can continue to function despite the loss of the ``leaders'', since their primary duty is to maintain the load balance.

In fact, the hard part about implementing a system like this is figuring out a protocol for doing load balancing. That is, from an abstract perspective, the system has the logical shape of a search tree (i.e. supergroups point to other supergroups), but in reality individual servers link to other individual servers. In the event that a server becomes overloaded, servers that link to that server should move their links to point to less loaded servers. In order to implement this there must be ways of discovering the underutilized servers of a given target supergroup (i.e. other servers in a given replication group or in a neighboring group in the same supergroup). In the event that all servers of a given supergroup are sufficiently loaded, new servers should be assigned to that supergroup, possibly after being removed from a less loaded supergroup.

4.3.2 Resolution in a Distributed System

The resolution process starts when a client sends a query containing a URN to be resolved to one of the participating servers. There may be a requirement that all clients who want access to the system do so through their own participating server, in order to push the cost of running the system out to the users. When a server receives a query, it first checks to see if the part of the database it is storing includes the URN in the query. If not, it routes the query to its parent in the search tree. If the URN in question is contained within the bounds of the database chunk, then that URN matches either a pointer to a server lower in the search tree or a record of namespace ownership. If the URN matches a pointer the query is routed according to that pointer.

If the URN falls into one of the URN spaces listed in the database chunk, then the owner of that URN has been determined. Next, the official hints provided by the owner of the namespace must be searched, hopefully determining which authoritative URN resolution service handles that URN. Typically, the amount of information provided by the owner is fairly limited, and is authenticated by the owner of the namespace and possibly by the root name authority. This information can also be augmented by unofficial hints, as is described in the next section.

The most apparently relevant hints (i.e. those relevant to the smallest URN space containing the URN in question) are then returned to the client. Depending on the name scheme, policy may require that official hints be authenticated by the root name registry. If there are unofficial hints as well, some of them may be authentically issued by the owner of the namespace (for example to correct a temporary outage), some may be authentically issued by the owner of the resource being named, and some may be completely unauthentic and misleading. The application will need to decide which hints to try first; for example, it may automatically use the official information by default, but raise a flag when other (possibly useful) information is available. If the user suspects that unofficial or inauthentic hints might be helpful, it should be easy to try them.

The servers in the distributed system use a protocol in which the state of the search is passed from one server to the next. When a query is received it is processed and either an answer is sent back to the client or the query is passed off to the next server in the search path. This passing process can be done with acknowledged UDP packets. This means that some state about the transaction must be kept from the time the query is received until the time the returning ACK is received.

4.3.3 Support for Unofficial Hint Information

It is advisable that these distribution systems incorporate the facility to store short-lived unofficial hint information along with the official hints from the name registry database for the following reasons:

1. The distributed systems and the official registry may be slow and/or expensive to change. Hence in many instances it may be desirable to inject hints directly into the various distributed resolution systems. The idea would be that injected hints are relatively fast-acting and cheap or free, but that the guaranteed quality of service is low. Probably a shared system can be constructed, in which users run a server in order to have an access point and that server is used in the distribution process. However, unlike the DNS, a user's personal information would not necessarily be stored on their node.

2. Topological variation can be implemented by injecting special localized hints that do not get copied onto topologically distant servers.

3. By providing an alternate pathway for information to be stored and served, the system becomes more tolerant of errors and outages in the registry. It also provides a way for publishers to fight denial of service; that is, if the registry revokes a publisher's authority or changes registered information without authorization, hints can be submitted which correct this. In order to make this work, authentication protocols must be implemented so that the hints can be verified to be produced by the publisher.[3]

It is important to remember that this hint information is a best-effort service. No guarantees are made about the reliability of the hint system. The policies governing hints, such as how much room there is to store hints, whether hints are allowed at all, whether hints cost money and whether hints require authentication, are all at the discretion of the designer of the distributed system. The only necessary agreement between designers is some kind of standard hint-passing protocol for disseminating new hints to the distributed systems that accept them.

In general, once a distributed system of the sort we describe has been implemented, hints are fairly easy to manage. Since the database is already distributed in the structure of a search tree, it is a simple operation to figure out on which servers a given hint belongs, by simply tracing the same route as a client's query. Once the right supergroup has been found, new hints can be distributed by a flooding protocol in a fashion similar to that in which updates to the database progress. Of course, hints are placed at a much lower priority in terms of consistency, but this suggests that they can be deployed more rapidly.

One of the important benefits of a hint system is that while it is less careful and less certain, it is much harder to subvert on a global scale. There is no centralized source for hints and no consistent distribution path. Interfering with a particular hint requires either gaining access to each individual server and deleting the hint, or finding a clever way to subvert the hint policy, such as flooding the system with bogus hints in the same part of the database. Hint policies, such as requiring payment for hint submission, can be constructed to make these tricks less appealing.

4.3.4 Conclusion

This concludes a vague sketch of the sort of distributed system we would need to serve a large flat namespace. Our objective here is not to lay out a detailed and polished design. We merely wish to suggest that something along these lines might have promise; we hope that whatever systems are put in place to answer the needs of the present include sufficient flexibility and extensibility to allow new and interesting systems to be tested.

5 A Wish List: Some Useful Additions to the NAPTR Proposal

Specifically, what kinds of extensibility would be most useful? If we are to assume that something like the NAPTR proposal is going to be implemented, it would be useful to add several points of extensibility to its design.

1. In addition to the current client protocol, a simple gateway protocol which sends the URN to a configured address and receives in return a collection of hint records in some format. These records might be similar to the SRV and NAPTR records, but would include additional fields such as owner, date, authentication, etc. The user could choose between the gateway protocol and the regular client protocol.

2. An authentication protocol could be built into the NAPTR system as well. Even if initially it is rarely used, by building parts of it in future security infrastructure can be integrated more easily.

3. The rewrite rules in the NAPTR system make it confusing. With unconstrained rules it quickly becomes difficult to specify and verify which URNs are whose property. One thing that would mitigate this problem would be to formulate a scenario for the typical or expected way that the system would be used (i.e. how the rules are set up). Another helpful piece of work would be the formulation of sample policies for restricting rewrite rules and for managing name authority servers.

6 Notes

[1] In some systems, semantics in the names is used during the initial attempt at resolution, and in cases where the semantics is found to be invalid a slower fallback method is used. While broken semantics will not cause the system to fail, such designs will degrade over time as an increasing fraction of the resolutions invoke the ``special'' case. If names are intended to last well beyond the lifetime of the semantic content, this type of design will not perform well in the long run.

[2] A more complex algorithm might do away with this requirement, but for simplicity we will assume it here.

[13]. A careful analysis of the issues involved with denial of service to publishers reveals a great deal of complexity. However, a system in which authenticated hints may be injected for free and limits are placed on the number produced by any one entity provides a reasonable level of protection.

7 References

[Sollins94] Sollins, K, and Masinter, L, Functional Requirements for Uniform Resource Names, RFC 1738, December 1994.

[Danzig] Danzig, P, DeLucia, D, Obraczka, K, Massively Replicating Services in Wide-Area Internetworks, Computer Science Dept., University of Southern California.

8 Contact Information

Lewis D. Girod
Research Programmer
(617) 253-3440
girod@lcs.mit.edu

Karen R. Sollins
Research Scientist
(617) 253-6006
sollins@lcs.mit.edu

Expires December 13, 1996